

USB 2.0 Interface

Daniel's inexpensive USB interface features an ADC0820 A/D converter chip and an EZ-USB interface. The system can sample 100,000 bps and send 512 bytes in 1 ms to a laptop.

The laptop industry has been gradually moving away from serial and parallel ports in favor of Universal Serial Bus (USB). In anticipation of the day when USB is the only available port for building interfaces to measure the external world, I developed an interface using National Semiconductor's inexpensive ADC0820 ADC chip and an EZ-USB interface.

The system can sample 100,000 sps and send 512 bytes in 1 ms to a laptop using Visual C++ 6.0 to display the voltage samples or a Fast Fourier transform (FFT). The FFT appears 50 times per second, which is sufficient for real-time frequency analysis. Cypress's EZ-USB kit is expensive, but it was the only one of the four systems that I tried that provided enough guidance and examples to develop my own application for the USB 2.0.

In this article I'll show you how to develop your own interfaces with USB. I'll also help you avoid several pitfalls.

WHY USB 2.0?

For my first USB project, I used an ActiveWire USB board to connect an ADC0820 to a 16-bit interface. It worked well, but because it was a USB 1.0 device, only 1 byte per millisecond was sent to the laptop. It took 0.512 s to send 512 bytes. The EZ-USB 2.0 interface, however, can send 512 bytes per millisecond, which speeds up the process considerably. It uses the bulk transfer method, which sends 512 bytes as a block of data. Note that it isn't a continuously sampling system, but it works well for FFT displays.

You can use a D-Link USB2.0 port CardBus if you don't have a USB 2.0 system installed and you have a CardBus interface (for PCMCIA cards). The card, which comes with a driver, works well, but it sometimes causes the laptop to stall at start-up when the

USB board isn't connected. You can shut down by disconnecting your laptop's AC power and battery. When you restart the computer, it will load the drivers successfully. Never panic.

EZ-USB

The secret is in the 8051 chip on the EZ-USB board. The EZ-USB 8051 architecture has numerous areas of memory called "endpoints" that you can use as storage buffers. Endpoint 8 is for data transfers. The key thing about endpoints is that the 8051 can fill them with the data it collects from your hardware. The USB software can access the endpoints and send the data to the laptop.

I built my 8051 program by modifying Cypress's example on the kit's CD-ROM. I then downloaded it via the control panel application, which was also on the CD-ROM.

The 8051 program takes the data from the ADC0820 and puts it into endpoint 8. When the endpoint 8 buffer is full, the EZ-USB board's USB software sends the data in the buffer to the laptop. The 8051 program then loops back to gather more data and fill the buffer.

The program in the laptop uses Visual C++. It incorporates a modified routine on the EZ-USB CD-ROM to request the data from the 8051.

When I tried to slow down the 8051 program to take fewer samples per second, the system appeared to suffer an interrupt. This occurred when the sample time was greater than about 28 ms. There's a way to get around this. Take the samples at a high rate and discard the unwanted samples. I chose a 20 ms sample time, which gave me 50,000 samples per second. That's 50 samples per millisecond, so it took roughly 10 ms to fill the 512-byte buffer.

The 8051 endpoints use double-buffering, which means you have to send the data to the endpoint twice, so the data is sent to the laptop approximately every 20 ms in a 1-ms time frame. This results in a display rate of 50 times per second.

The C++ display screen enables you to take a block of 512 data points and display data or step through each value using the index of the data array. The other mode is used for entering the number of cycles (each cycle is a block of 512 data points) and displaying the FFT. This particular application was set up to display frequencies up to 2.5 kHz, which is good for showing voice frequencies, guitar frequencies, and signals coming from a ham radio code filter. I created the scale on the display by feeding in known frequencies from an audio signal generator and literally writing in the appropriate scale. With a sampling frequency of 100 kHz, you could have a scale that goes to 50 kHz (the Nyquist limit).

APPROACH THE PROBLEM

If you're interested in USB, you've probably spent some time researching the subject. By reading books like Jan Axelson's *USB Complete* and John Hyde's *USB Design By Example*, I learned that I needed the bulk transfer data transfer method, which transfers a block of data (512 bytes in this case) from the 8051 endpoint to the laptop. C++ receives the data in an array.

The standard procedure is to get a known application running by way of programs from the EZ-USB CD-ROM. If the gods are kind to you, the source code will be there too. After you find a program similar to your application, you can begin modifying it. I focused on a program for the 8051 and a program for the laptop.

In typical C and C++ programs, there are several files that you must compile and link to make the executable file. You should use programs from the EZ-USB CD-ROM that have source code files. Typically, there's a subdirectory for each example that includes all the associated programs.

HACK THE CODE

Let's start with the program for the 8051. You can test your program with the Cypress control panel, which you

can install with the EZ-USB CD-ROM. Mind you, this isn't the control panel furnished by Windows XP. The CD-ROM installs the control panel under the Cypress entry that appears when you click the "All Programs" bar in the Windows Start box.

The 8051 program is written in C and uses the Keil compiler on the EZ-USB CD-ROM. Install the compiler during the set-up process. After the program is compiled, download the executable file to the 8051 using the control panel. You

can also download the program to an EEPROM on the board. Typical applications have two C files that are compiled with other files to make the 8051 executable file. Listing 1 shows the code added to the fw.c file to take data from the ADC0820 and store it in the buffer.

If you install the EZ-USB files, the original fw.c file for the application will be located in C:\Cypress\USB\examples\FX2\bulksrc. I copied this entire directory to the FX2APPS directory, where I modified the fw.c file. The bulksrc directory contains the files needed to form the executable file. The file became ADCDATA8 after approximately 30 iterations.

The EZ-USB kit also includes "EZ-USB FX2 CY7C68013 Technical Reference Manual" (Cypress, 2001), which explains how to use output enable port B (OEB), output enable port D (OED), and send data to port B (IOB). I used port B for control signals to and from the ADC0820. I used port D to get the data from the ADC0820.

The EZ-USB board is double-buffered, so you must send the ADC data (512 bytes) to endpoint 8 twice. I placed the code right after the TD poll command within the main program built into the fw.c program.

Figure 1 (p. 32) is the wiring diagram I used to connect the ADC0820 to the 8051. A 7805 voltage regulator supplies 5 V for the chip. The large-value electrolytics help remove noise from the power supply. You should also put 0.1- μ F capacitors at all the 5-V inputs to the chip.

The EZ-USB kit comes with an expansion board that connects to the main board via headers J1 through J6. I used only J1 because it contains both ports. A 34-pin connector (often used to connect floppy and hard drives to a motherboard) connects with J1. Photo 1 (p. 32) shows the ADC circuit and its connections to the EZ-USB board.

Port B bits are used for the start pulse (B0), output enable (B1), and INT (B2) for the ADC0820. The start pulse initiates the conversion process. The INT pulse lets the 8051 know that the conversion is complete. The output enable pulse enables the ADC0820's tristate outputs. The data from the ADC0820 output pins is read from port D (D0 through D7) and stored in the EP8FIFOBUF array. The

Listing 1—I used this C code for the 8051. The routine was inserted in the fw.c program to load data from the ADC0820 to the 8051 endpoint 8 buffer. The system is double-buffered, and the data must be entered twice.

```
TD_Poll();           //This command is already in the main "while loop."
REVCTL = 0x03;       //The code starts here.
SYNCDelay;          //This is a time delay.
FIFORESET = 0x80;    //Reset buffer
SYNCDelay;
FIFORESET = 0x08;
SYNCDelay;
FIFORESET = 0x00;
SYNCDelay;
EP8CFG = 0xE0;       //Enable endpoint 8
OED = 0x00;          //Make all of port D as input bits
OEB = 0x03;          //Make two lowest bits of port B the output bits
                    //The remaining 6 bits are input bits
for (i=0; i<512; i++) //Loop 512 times
{
    IOB = 0x03;       //B0 is active low
    IOB = 0x03;
    ReadyLoop: JSTATUS = IOB; //Read port B
    while (JSTATUS & 0x04) //Keep looping if bit B2 = 1.
        goto ReadyLoop;
    IOB = 0x01;       //Send ADC output enable command to bit B1.
    JDATA = IOD;      //Read data from port D
    EP8FIFOBUF[i] = JDATA;
    L=1;              //Delay loop to slow down sampling frequency.
    for (k=0; k<5; k++)
    { L=L+1;
    }
}
SYNCDelay;
EP8BCH = 0x02;       //Enter number of bytes in buffer (high byte).
SYNCDelay;
EP8BCL = 0x00;       //Enter number of bytes in buffer (low byte).
for (i=0; i<512; i++) //Do it all again for second buffer.
{
    IOB = 0x03;
    IOB = 0x02;
    IOB = 0x03;
    ReadyLoop2: JSTATUS = IOB;
    while (JSTATUS & 0x04)
        goto ReadyLoop2;
    IOB = 0x01;
    JDATA = IOD;
    EP8FIFOBUF[i] = JDATA;
    L=1;              //Delay Loop to slow down sampling frequency
    for (k=0; k<5; k++)
    { L=L+1;
    }
}
SYNCDelay;
EP8BCH = 0x02;
SYNCDelay;
EP8BCL = 0x00;       //Code ends here
```

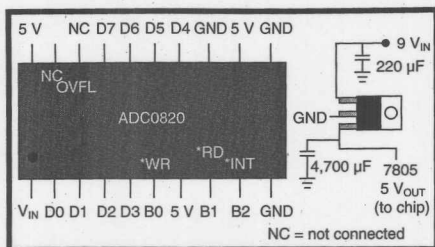


Figure 1—The ADC connections run to the 8051's ports D and B. D0 through D7 are the ADC0820's data output pins. B0 starts the data conversion. B1 enables the data output pins. B2 tells the 8051 that the data is ready.

routine loops to gather 512 data bytes. Finally, the buffer is declared full by putting the buffer count (512 bytes) in EP8BCH and EP8BCL.

When the buffer is declared full, the routine can't write to the buffer. The program simply skips over the EP8FIF0BUFF[i]=JDATA command and continues on. It then loops through the main while loop until the buffers become available again.

You can use the control panel to make sure the 8051 program works correctly. The numbers from the endpoint buffer represent actual voltage levels, so you can feed in different DC levels and test

the ADC. The control panel also enables you to transfer data from the endpoint to the display screen by using the "bulk int" button. Just make sure you replace the string length (64) with the new length (512). Also, you need to change the number of bytes to 512. When you're satisfied that the 8051 program is running correctly, it will be time to jump into the next phase, which involves developing the laptop program.

BACK AT THE LAPTOP...

The laptop program is based on the BulkXfer program located in C:\Cypress\USB\Examples\EzUsb\BulkXfer. Again, I copied the entire directory into a new directory (C:\adc_app). The C++ dialog file BulkXferDlg.cpp was modified to collect the data from the EP8FIF0BUFF array and store it in another array (j_adcdata). Listing 2 shows the routine that requests the data from the buffer and puts it into your array. After the data is in this array, you can use it for the various functions.

I discovered the correct pipe number by checking the 8051 software with

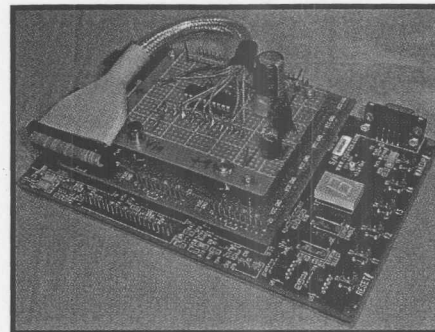


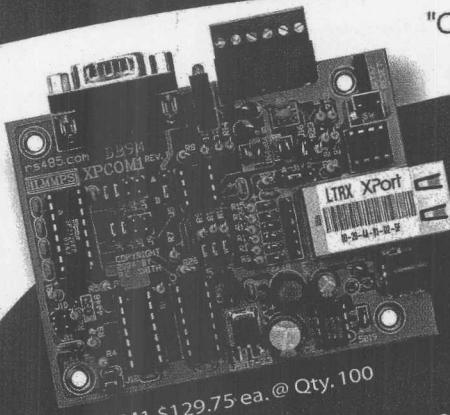
Photo 1—The EZ-USB board is on the bottom. The expansion board fits over headers J1 through J6. The ADC board is fastened to the expansion board with 0.375" spacers so you can easily remove it for your next project.

the Cypress control panel. Using the Get Pipes button, I found that endpoint 8 was attached to pipe 3.

I modified the BulkXfer dialog screen to include the functions I wanted for the FFT and data displays. Obviously, the BulkXferDlg.cpp file was extensively edited, which is why it isn't listed completely. All the files for the modified bulksrc and BulkXfer programs are posted on the *Circuit Cellar* web site.

Photo 2a shows a typical data display. Photo 2b shows an FFT display. The data

"Communications Glue" ABC<=>XYZ



XPCOM1 \$129.75 ea. @ Qty. 100

Download our free Coded RS232/485 & Ethernet Terminal Emulators from www.rs485.com

Call the RS485/Ethernet/Interface/Communications Wizards @ 513 874 4796 Today!

- RS232/RS485/RS422/TTL
- USB/900 MHz Radio
- Converters/Repeaters
- Fiber Optics/Isolators
- Multi-Repeaters/Hubs
- Extenders/Multidrop
- 10/100 Base-T Ethernet
- Microprocessor Serial I/O
- Code Activated Switches



Extensive Interface Product Line

Custom Units & Smart Units
Digital/Relay I/O & "Extension Cords"
Vending/Security status over TCP/IP
5-Axis Camera PTZ, Focus, Iris Control
Large Multidrop & "Home Run" RS485
Card Reader Electronics & Control
VF/LCD Display Remote Control
"Industrial units" & -40C to 85C
3.0KV Isolation & wide Power range

RESmith
WWW.RS485.COM
4311 R.E. Smith Drive, Hamilton, Ohio 45011

513-874-4796

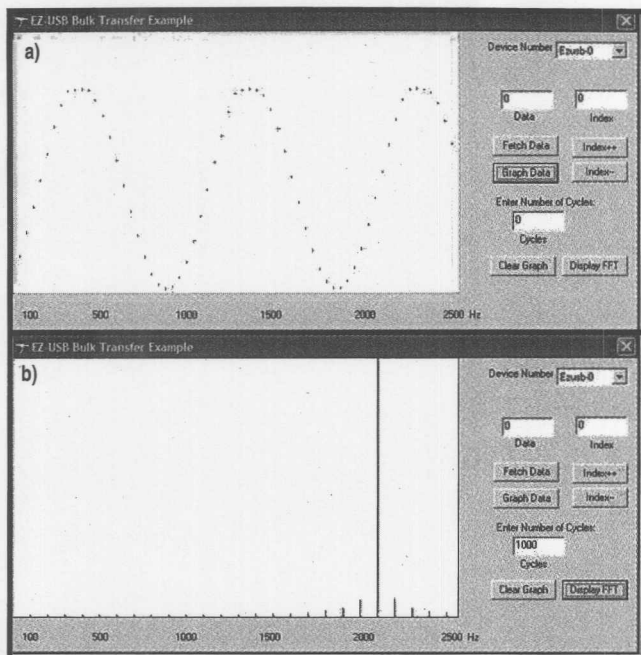


Photo 2a—The display shows the first 64 bytes of a whistle sound input. Use the Index buttons to find additional bytes. The frequency scale is used only for the FFT display. The sample rate is 50 kHz. **b**—Take a look at one frame of the FFT routine. The system displays 50 frames per second, which enables you to display data in real time. The frequency scale shows the main components of the whistle, which is slightly lower in tone than the whistle in Photo 2a.

comes from a single 512-byte block that's loaded after you click the Fetch Data button. The data is displayed after you click the Graph Data button. It's a good idea to click the Clear Graph button before taking the data because it'll give you a solid black background. Use the index buttons to inspect the data values one by one.

The FFT resolution was approximately 100 Hz, which was adequate for my work. I took the FFT routine, which uses the BASIC language, from Hal Chamberlin's book, *Musical Applications of Microprocessors*. I converted it to C++ for this application. The screen is updated 50 times

per second, so I get a 1-min. count by entering 3000 in the Cycles box.

program into the EEPROM. Because the USB software depends on the various identification bytes to choose the correct driver, changing the EEPROM could change the bytes. If you're lucky, the CD-ROM will contain another driver that will connect with your board when the first driver stops working. This happened to me. Now I have to load the monitor program manually. It used to load automatically at start-up.

When I attempted to compile the BulkXfer file, I received a message stating that the include file devioctl.h couldn't be opened. This is a Windows device I/O control header file that wasn't included

OH YE PITFALLS

Make backups of the working directories. Doing so will keep you from weeping if Visual C++ does something to your files. Be careful about erasing portions of an EZ-USB file you're modifying for your application. It's easy to erase too much code. It's much better to use forward slashes (/) to turn the line into a remark.

You also should be extremely careful when loading a

in the EZ-USB software. You can find it in the Windows device developer's kit (WINDDK). The free DDK is included with the other files for this project.

APPLICABILITY

Setting up the USB port for A/D conversion enables a host of experiments. I use the system to acquaint my students with the basics of digital signal processing and digital filters. They can see the results of digitizing an analog signal as well as an immediate application in the FFT. Watching the frequencies change as they apply various inputs (signal generators, whistles, voices), students develop an intuitive understanding of the process. It's also a concrete example of how they can use Visual C++ for data manipulation and display. But, most of all, it shows them that it can be done. 📌

Daniel Cross-Cole worked for the Department of Navy for 22.5 years as an electronics engineer. He retired in 2003 and is now a professor at DeVry University in Arlington, Virginia. Dan currently teaches courses about digital signal processing, telecommunications engineering, and computer peripheral devices. He enjoys ham radio and using Fast Fourier Transforms to characterize acoustic guitars. You may contact him at DCrossCole@dc.devry.edu.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2005/178.

RESOURCES

J. Axelson, *USB Complete*, Lakeview Research, Madison, WI, 2001.

H. Chamberlin's, *Musical Applications of Microprocessors*, Hayden Book Co., Rochelle Park, NJ, 1985.

J. Hyde, *USB Design By Example*, Intel Press, Santa Clara, CA, 2001.

SOURCES

EZ-USB Kit

Cypress Semiconductor Corp.
www.cypress.com

ADC0820 A/D converter chip

National Semiconductor
www.national.com

Listing 2—I used this routine to get data from the 8051's endpoint 8. Endpoint 8 data was transferred via pipe 3. The bulk transfer put the data into `j_adcddata`. I then transferred it to a buffer named "array."

```
btc.pipenum = 3;
// Read data in from EZUSB device
bResult = DeviceIoControl(hDevice,
    IOCTL_EZUSB_BULK_READ,
    &btc,
    sizeof
    (BULK_TRANSFER_CONTROL),
    j_adcddata,
    512,
    (unsigned long *)&nBytes,
    NULL);
for (i = 1; i < 513; i++)
    Array[i]=j_adcddata[i];
```